

# Caches II

CSE 351 Winter 2019



## Instructors:

Max Willsey

Luis Ceze

## Teaching Assistants:

Britt Henderson

Lukas Joswiak

Josie Lee

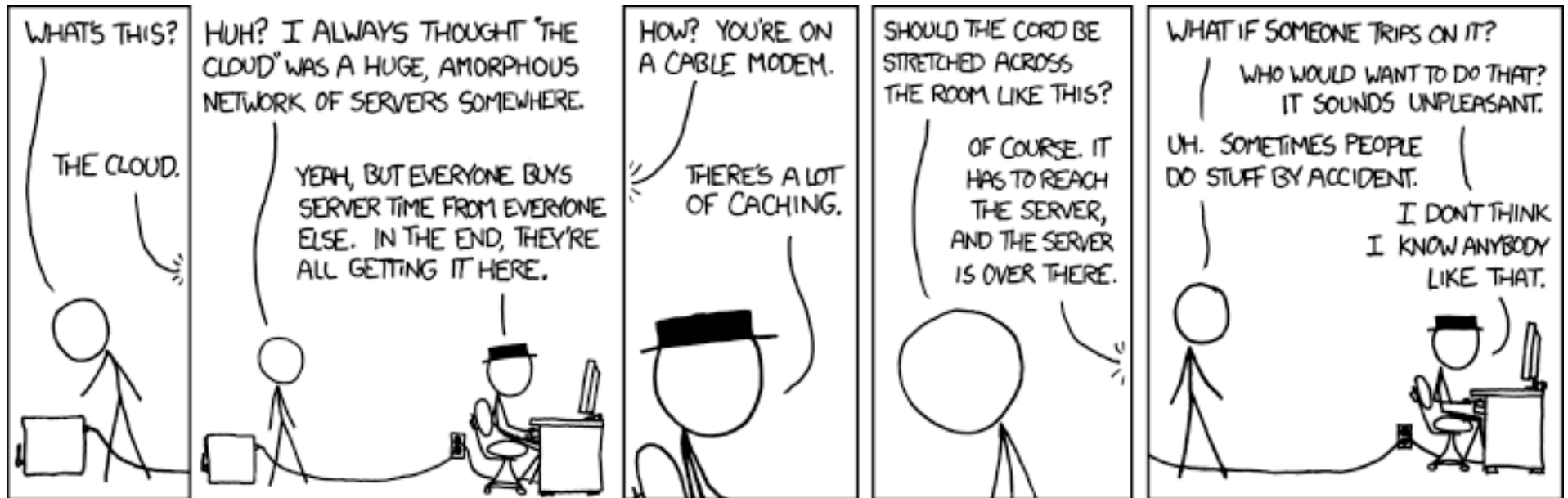
Wei Lin

Daniel Snitkovsky

Luis Vega

Kory Watson

Ivy Yu

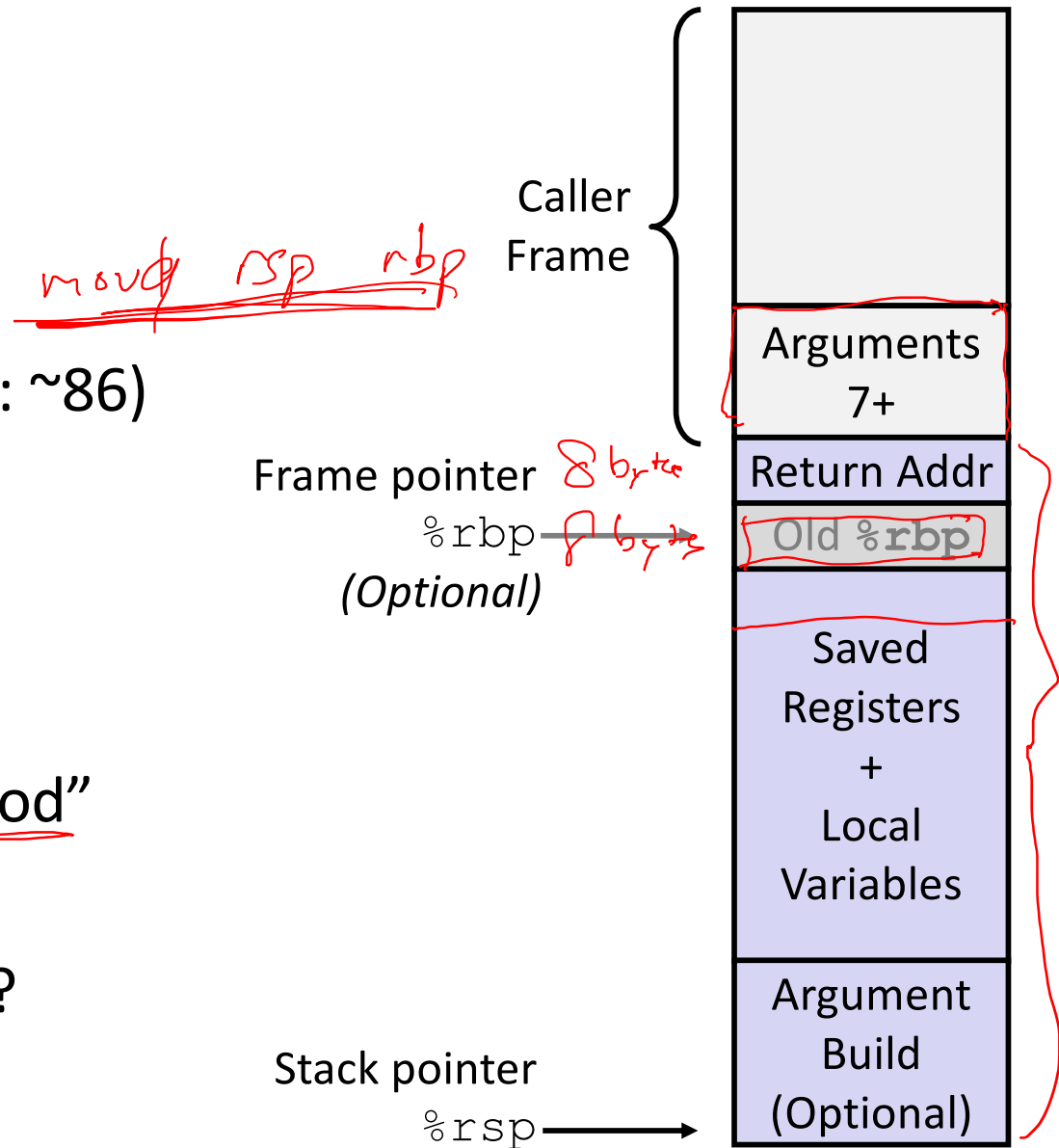


# Administrivia

- ❖ Lab 3 due Friday (02/22)
- ❖ **Mid-Quarter Survey Feedback**
  - Pace is “too slow” to “too fast”
  - Use office hours!
  - I’ll try to post ink when iPad doesn’t eat it

# Midterm

- ❖ Grades
  - Coming out soon
  - Did really well! (mean: ~86)
  - Final will be harder
- ❖ Regrade requests
- ❖ Common things
  - “function”, not “method”
  - tail recursion
  - What is a stack frame?



# Last time

## ❖ Caching in general

- Successively higher levels contain “most used” data from lower levels
- Exploits *temporal and spatial locality*
- Caches are intermediate storage levels used to optimize data transfers between any system elements with different characteristics

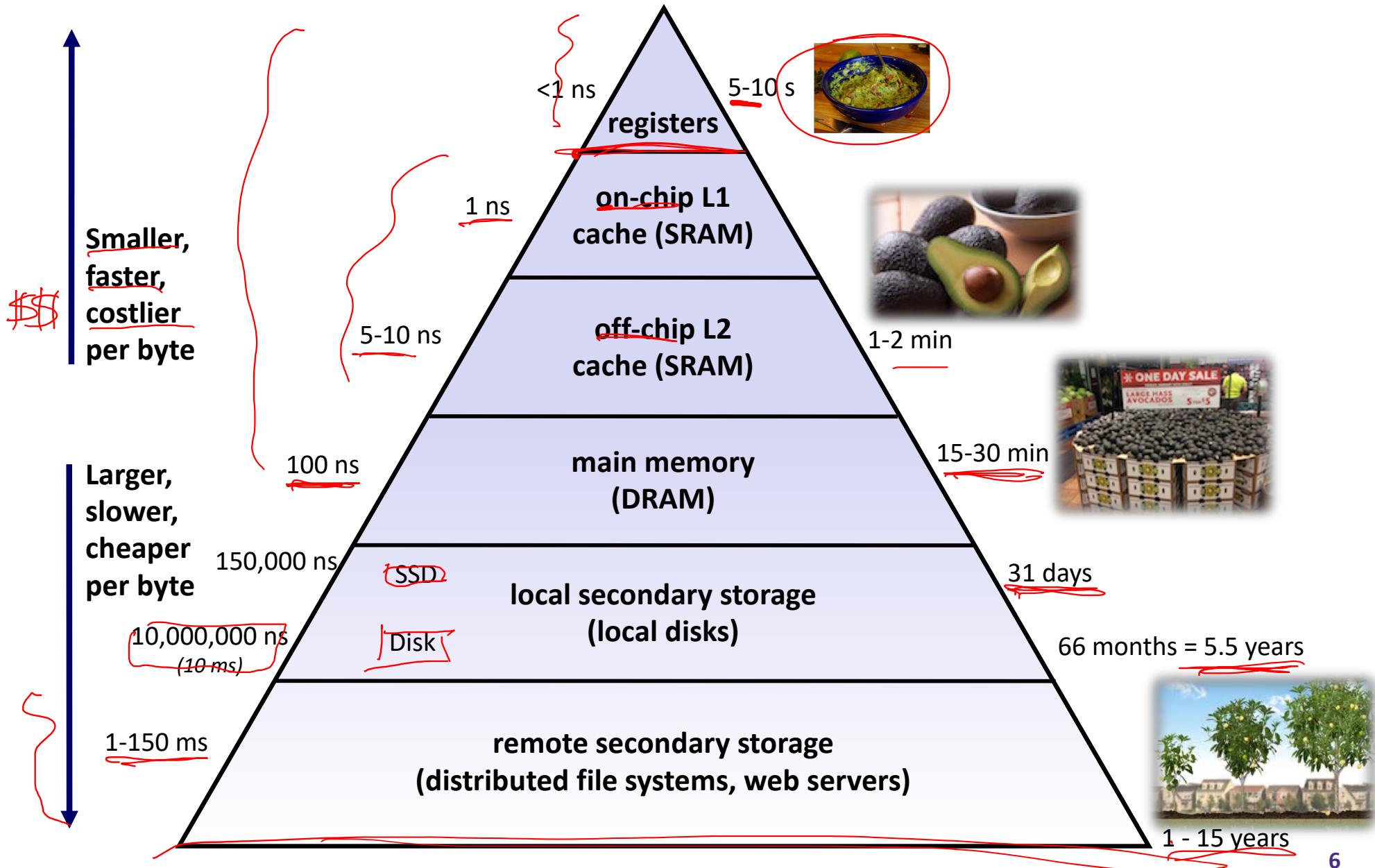
## ❖ Cache Performance

- Ideal case: found in cache (hit)
- Bad case: not found in cache (miss), search in next level
- Average Memory Access Time (AMAT) = HT + MR × MP
  - Hurt by Miss Rate and Miss Penalty

# Can we have more than one cache?

- ❖ Why would we want to do that?
  - Avoid going to memory!
- ❖ Typical performance numbers:
  - Miss Rate
    - L1 MR = 3-10%
    - L2 MR = Quite small (*e.g.* < 1%), depending on parameters, etc.
  - Hit Time
    - L1 HT = 4 clock cycles
    - L2 HT = 10 clock cycles
  - Miss Penalty
    - P = 50-200 cycles for missing in L2 & going to main memory
    - Trend: increasing!

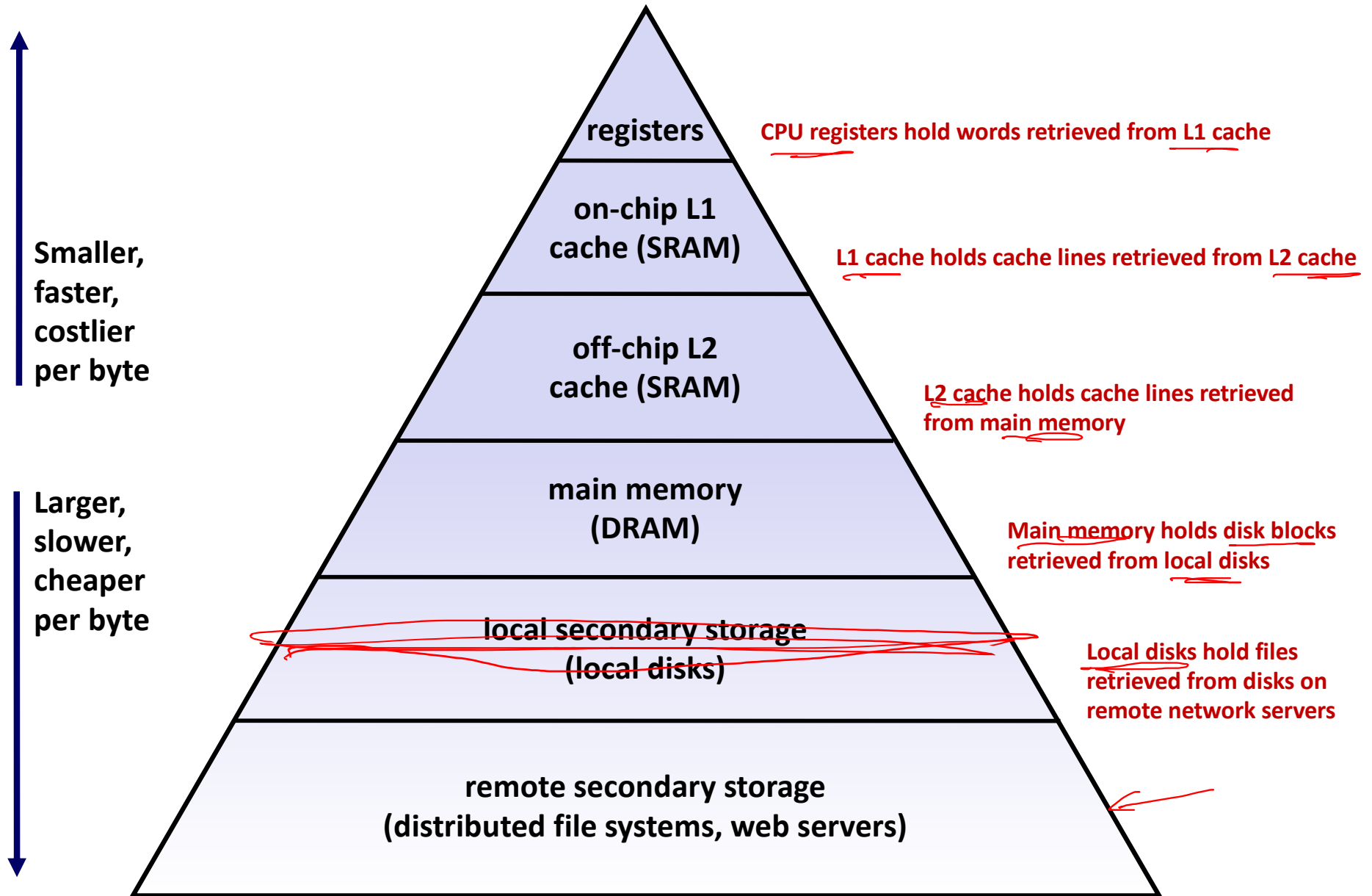
# An Example Memory Hierarchy



# Memory Hierarchies

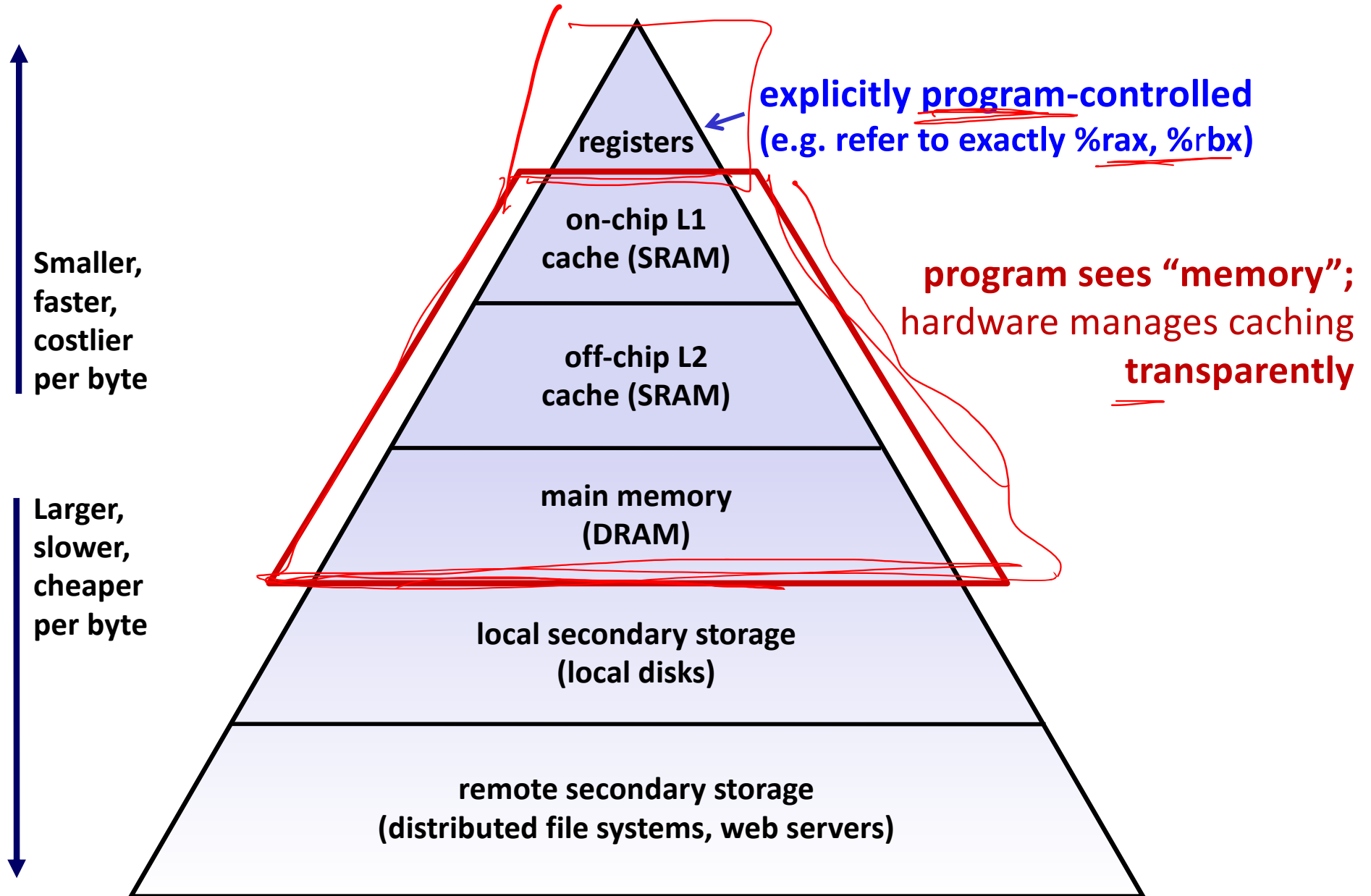
- ❖ Some fundamental and enduring properties of hardware and software systems:
  - Faster = smaller = more expensive
  - Slower = bigger = cheaper
  - The gaps between memory technology speeds are widening
    - True for: registers  $\leftrightarrow$  cache, cache  $\leftrightarrow$  DRAM, DRAM  $\leftrightarrow$  disk, etc.
  - Well-written programs tend to exhibit good locality
- ❖ These properties complement each other beautifully
  - They suggest an approach for organizing memory and storage systems known as a memory hierarchy
    - For each level  $k$ , the faster, smaller device at level  $k$  serves as a cache for the larger, slower device at level  $k+1$

# An Example Memory Hierarchy



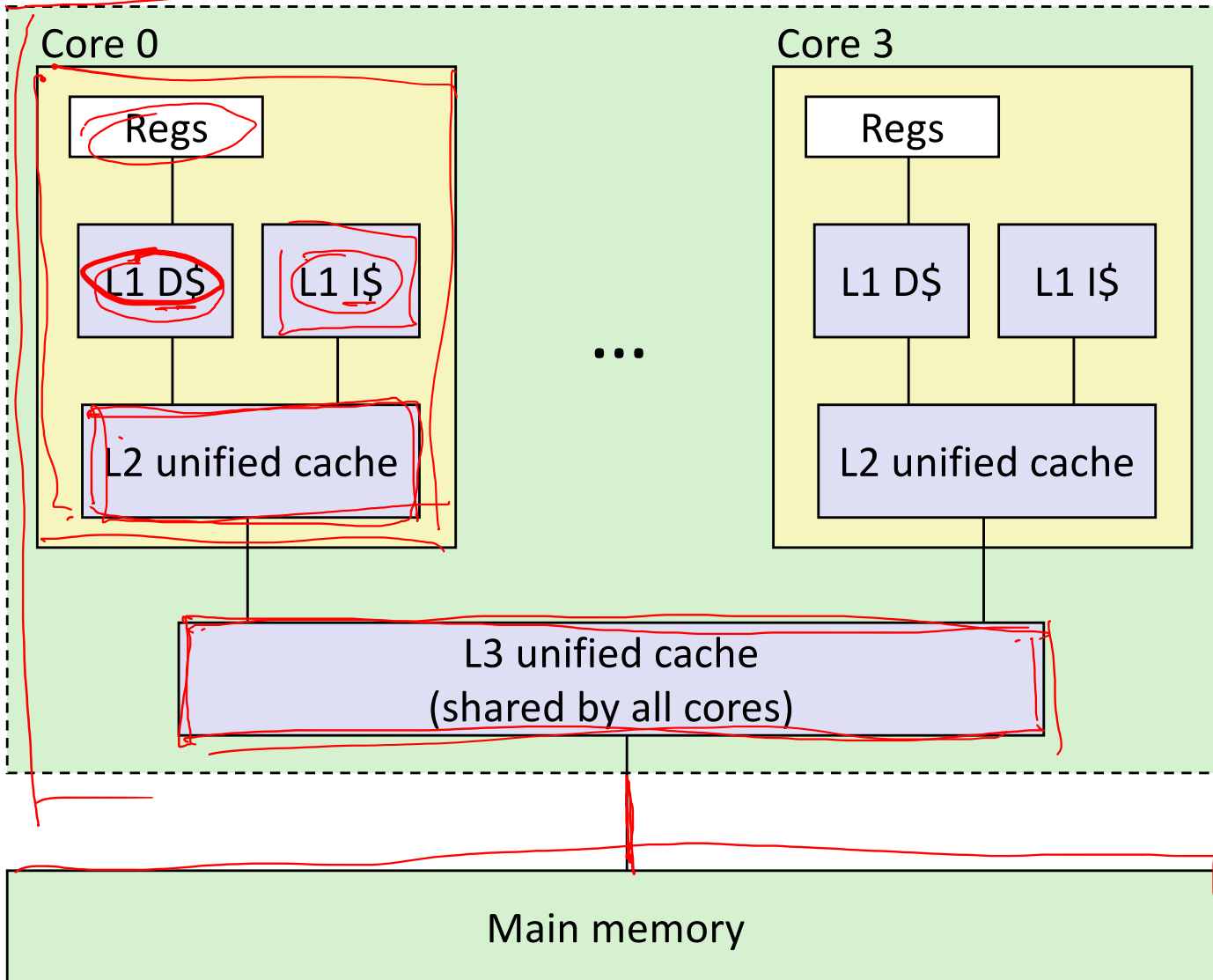


# An Example Memory Hierarchy



# Intel Core i7 Cache Hierarchy

Processor package



Block size:

64 bytes for all caches

L1 i-cache and d-cache:

32 KiB, 8-way,  
Access: 4 cycles

L2 unified cache:


256 KiB, 8-way,  
Access: 11 cycles

L3 unified cache:

8 MiB, 16-way,  
Access: 30-40 cycles

3-500 cycles

# Making memory accesses fast!

- ❖ Cache basics
  - ❖ Principle of locality
  - ❖ Memory hierarchies
  - ❖ **Cache organization**
    - **Direct-mapped (*sets*; index + tag)**
    - **Associativity (*ways*)**
    - Replacement policy
    - Handling writes
  - ❖ Program optimizations that consider caches
- 

# Cache Organization (1)

**Note:** The textbook uses “B” for block size

- ❖ **Block Size ( $K$ ):** unit of transfer between \$ and Mem
  - Given in bytes and always a power of 2 (*e.g.* 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!

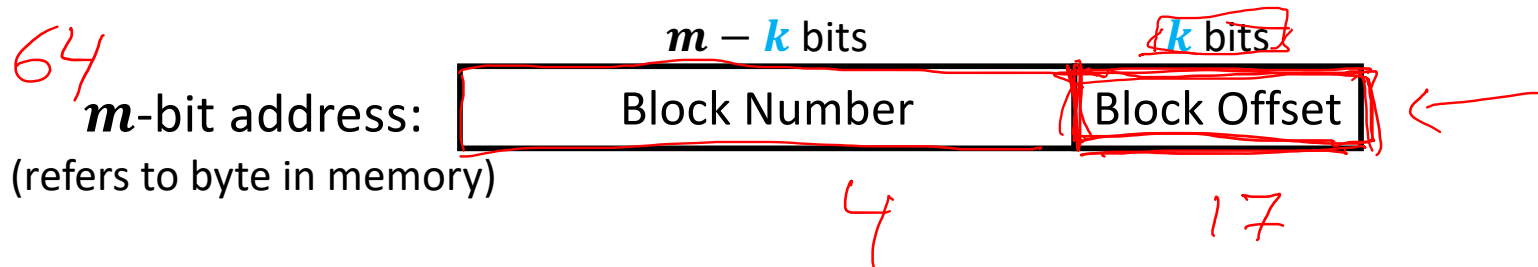
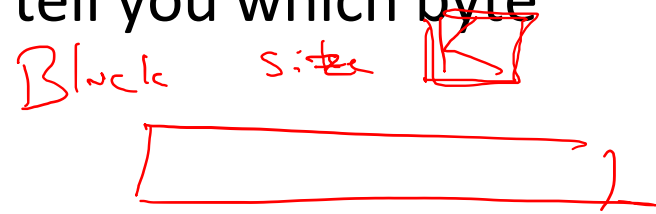
# Cache Organization (1)

**Note:** The textbook uses "b" for offset bits

- ❖ **Block Size ( $K$ ):** unit of transfer between \$ and Mem
  - Given in bytes and always a power of 2 (e.g. 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!

## ❖ Offset field

- Low-order  $\log_2(K) = k$  bits of address tell you which byte within a block
  - (address) mod  $2^n = n$  lowest bits of address
- (address) modulo (# of bytes in a block)

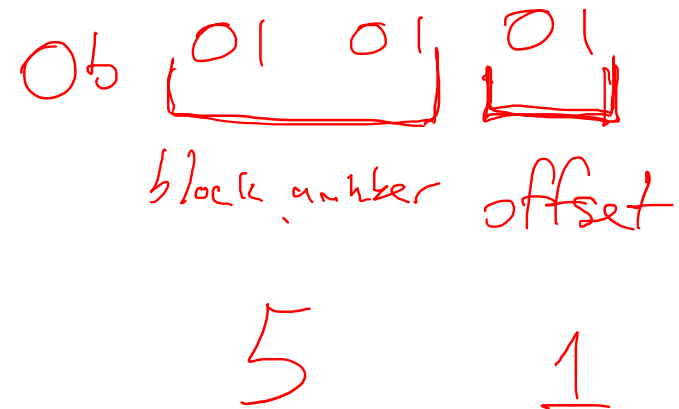


# Peer Instruction Question

$$k = \log_2 K = 2$$

❖ If we have 6-bit addresses and block size  $K = 4$  B, which block and byte does 0x15 refer to?

- |    | Block Num   | Block Offset  |
|----|---|---|
| A. | <u>1</u>  | <u>1</u>  |
| B. | <u>1</u>  | <u>5</u>  |
| C. | <span style="border: 1px solid red; padding: 2px;">5</span> | <span style="border: 1px solid red; padding: 2px;">1</span> |
| D. | <u>5</u>  | <u>5</u>  |
| E. | We're lost...   |   |

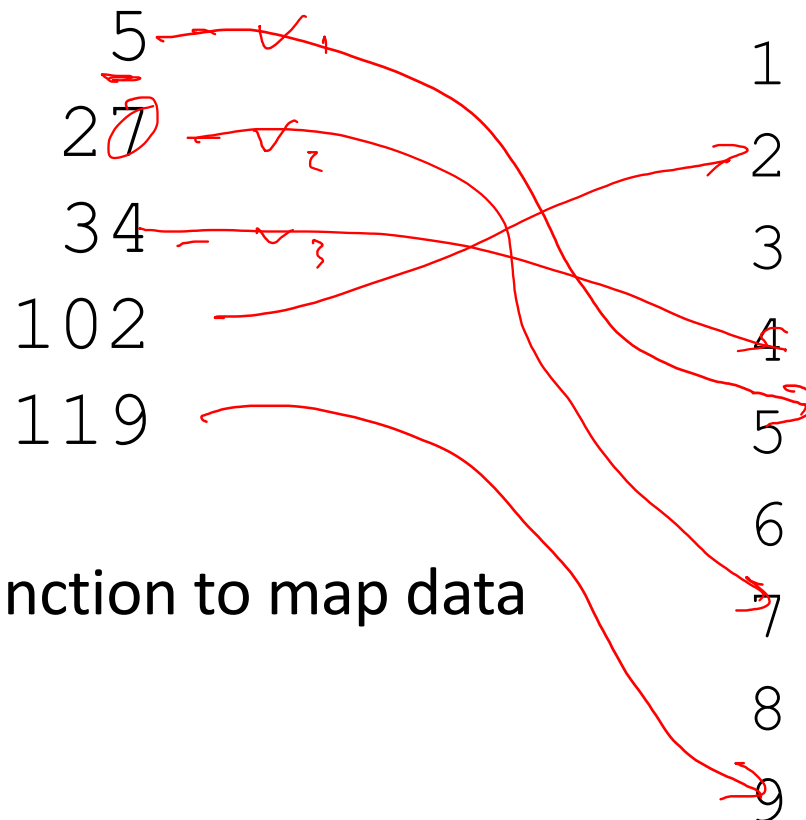


## Cache Organization (2)

- ❖ **Cache Size ( $C$ )**: amount of *data* the  $C$  can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes ( $C$ ) or number of blocks ( $C/K$ )
  - Example:  $C = 32 \text{ KiB} = 512$  blocks if using 64-B blocks
- ❖ Where should data go in the cache?
  - We need a mapping from memory addresses to specific locations in the cache to make checking the cache for an address **fast**
- ❖ What is a data structure that provides fast lookup?
  - Hash table!

# Review: Hash Tables for Fast Lookup

Insert:



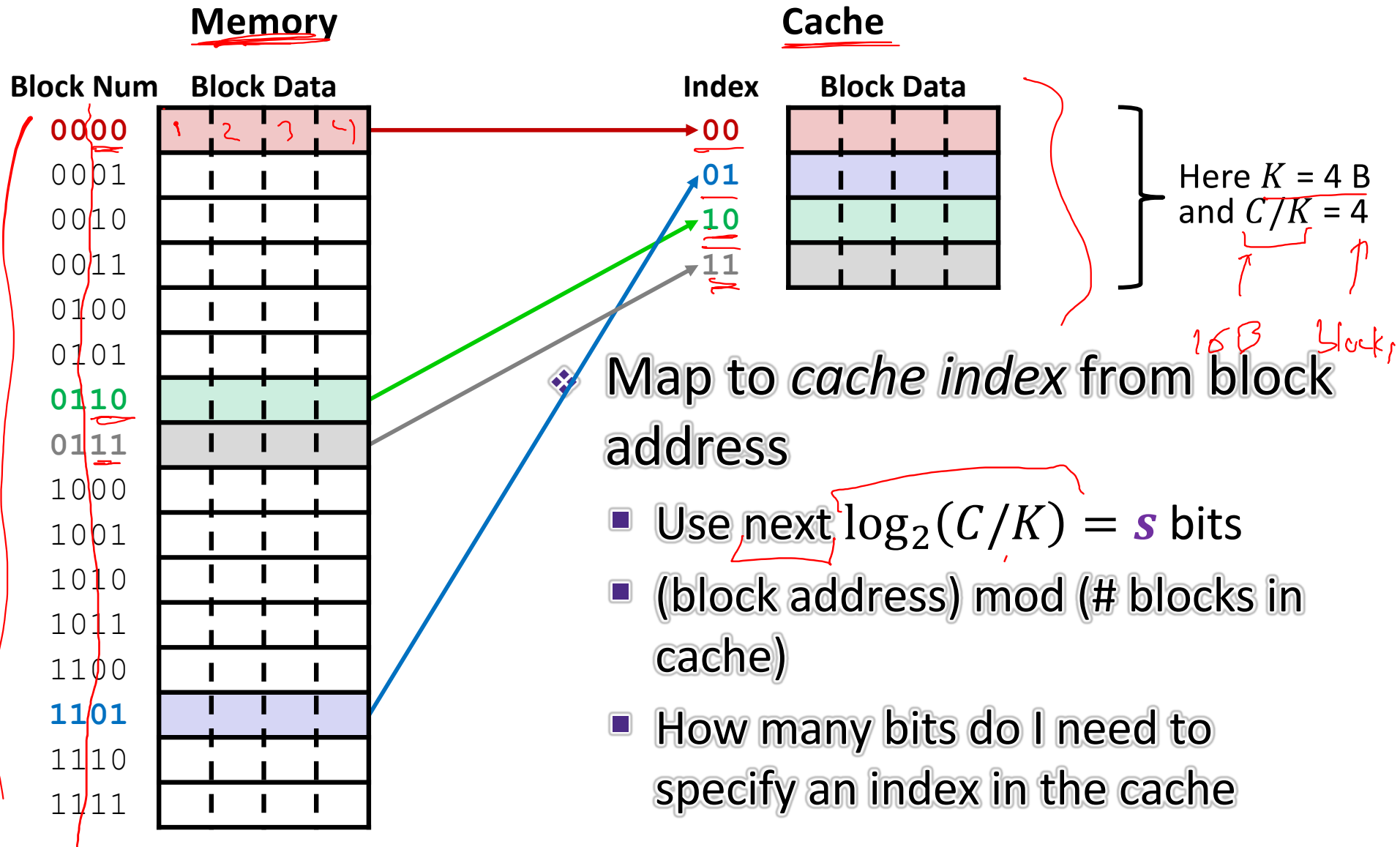
0
1
2
3
4
5
6
7
8
9

Apply hash function to map data to "buckets"



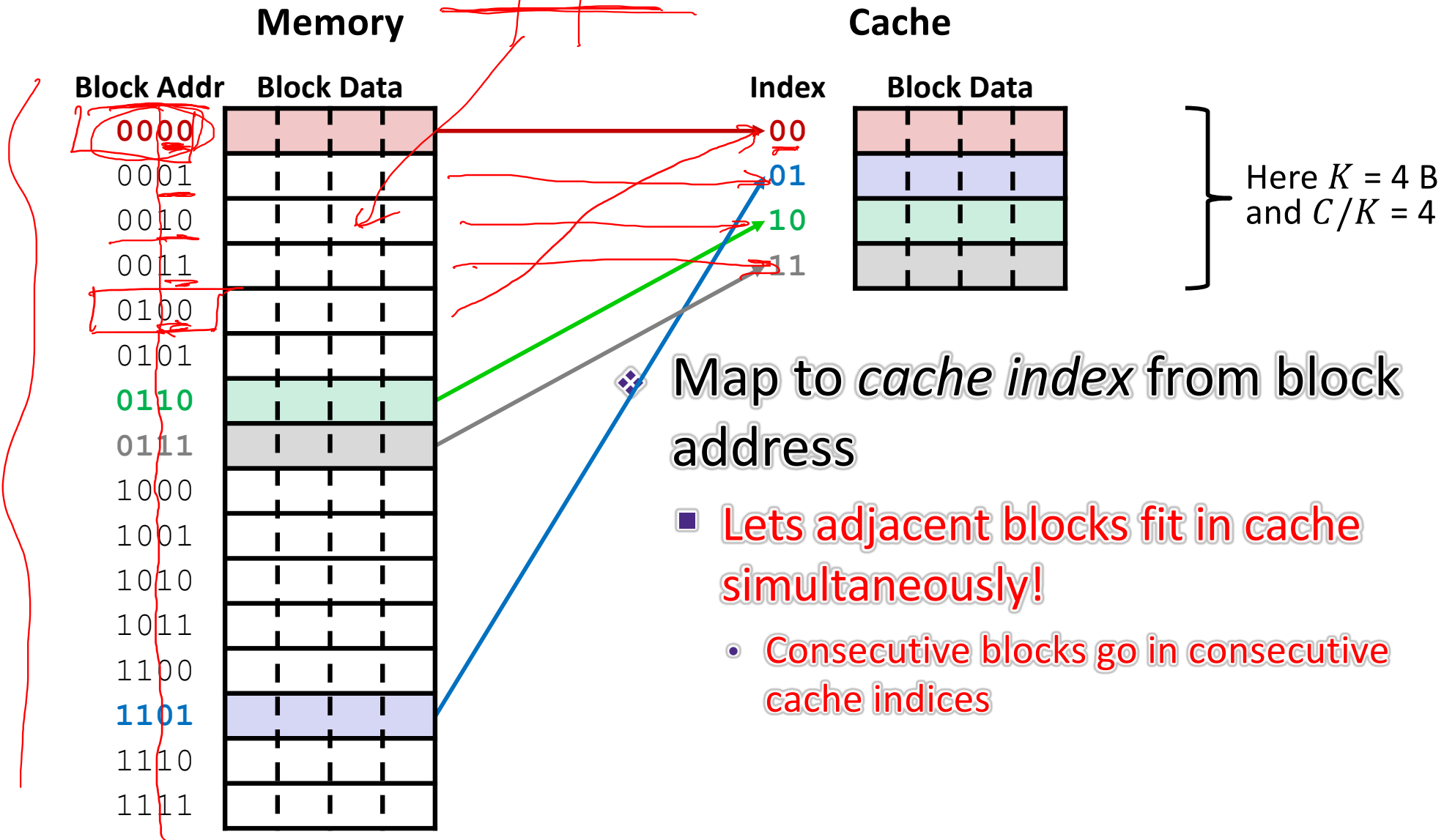
# Place Data in Cache by Hashing Address

0b 0000 00



# Place Data in Cache by Hashing Address

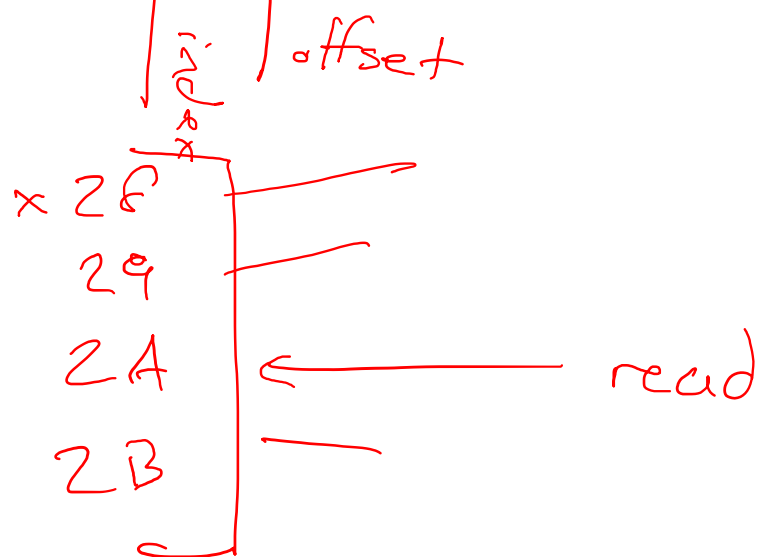
*index*  
*offset*  
 00 | 010



# Practice Question

- ❖ 6-bit addresses, block size  $K = 4$  B, and our cache holds  $S = 4$  blocks.
- ❖ A request for address 0x2A results in a cache miss. Which index does this block get loaded into and which 3 other addresses are loaded along with it?

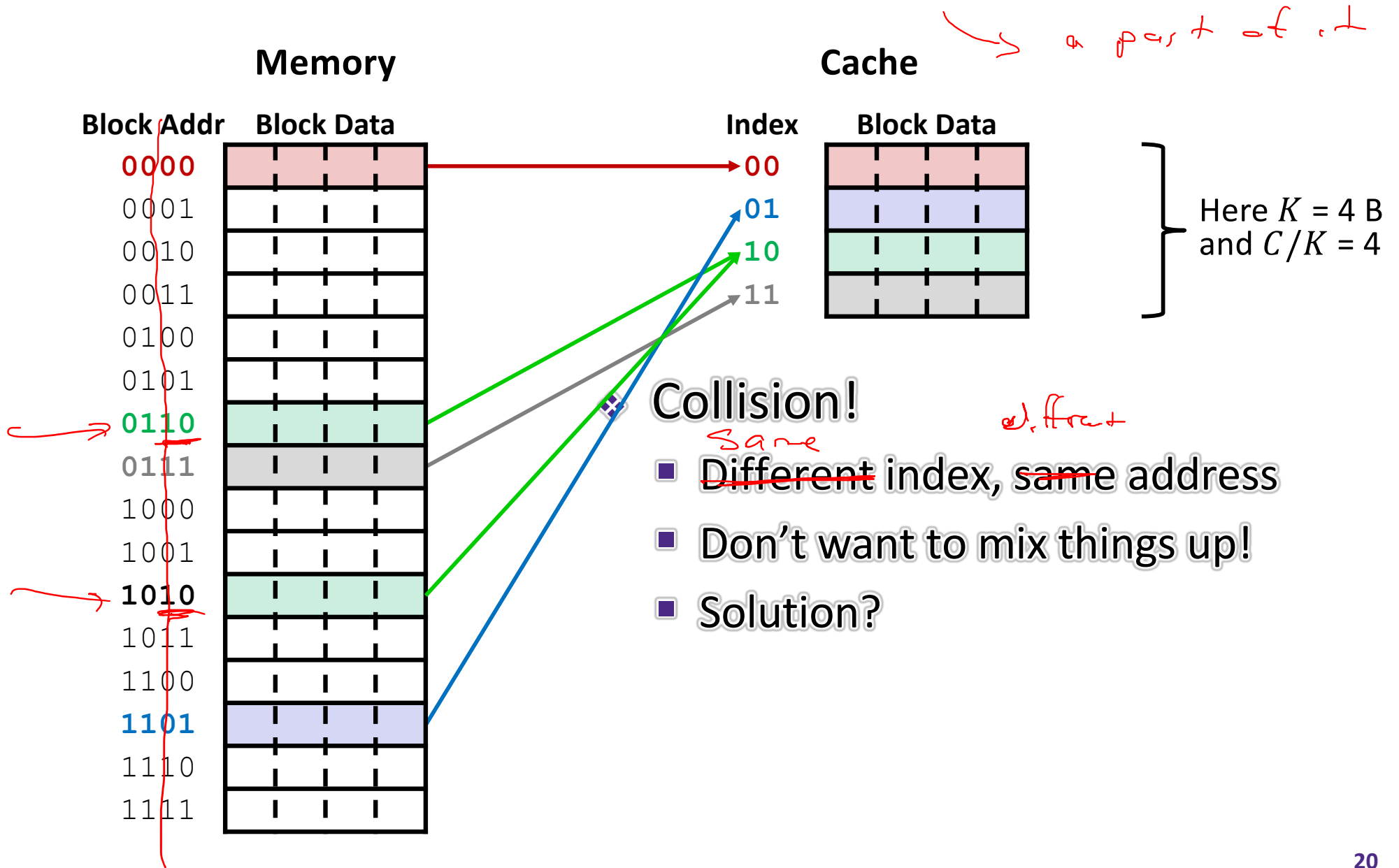
$0x2A = 0b101010$



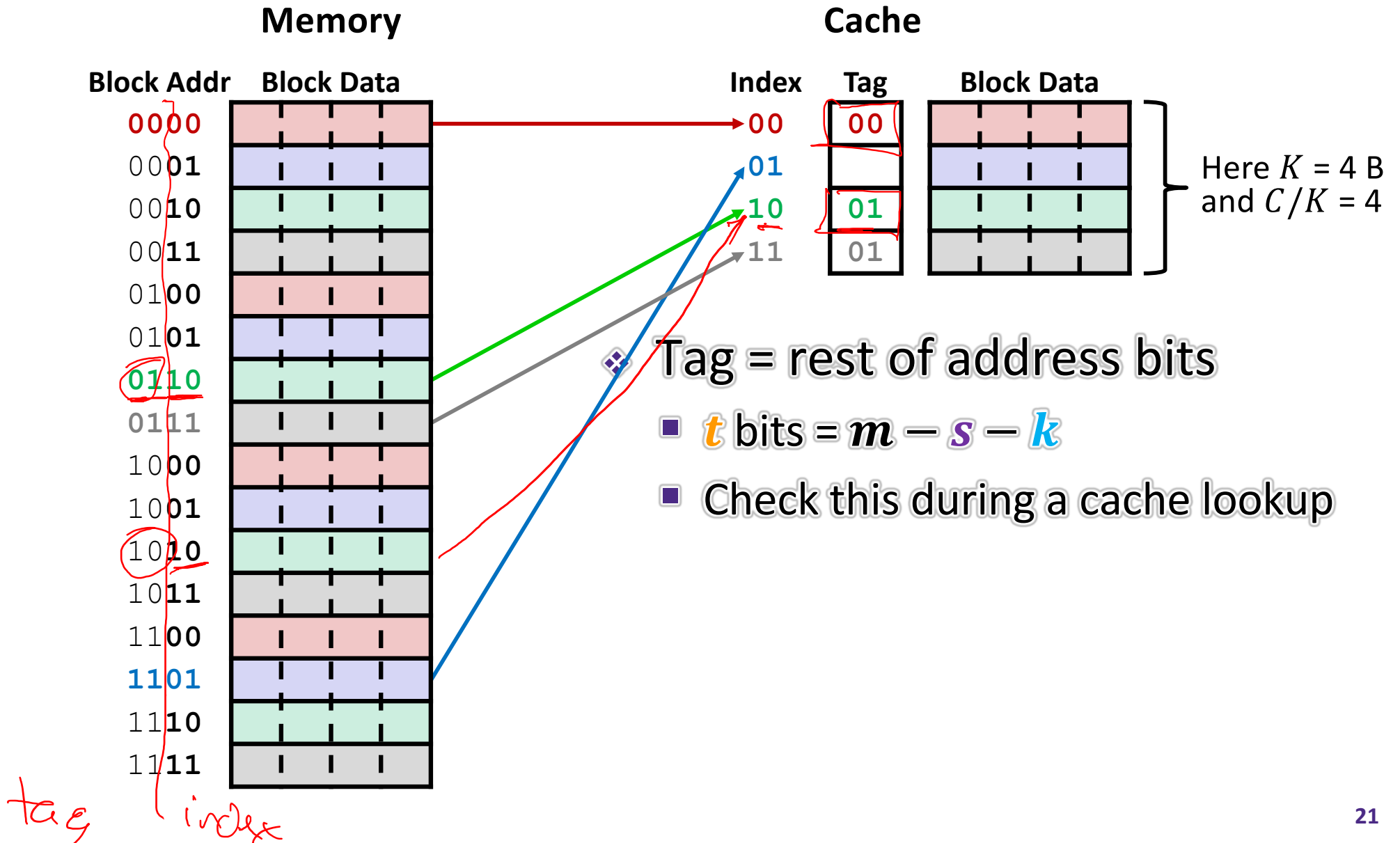
index = 2

brnum  
 1010 00  
 01  
 10  
 11

# Place Data in Cache by Hashing Address



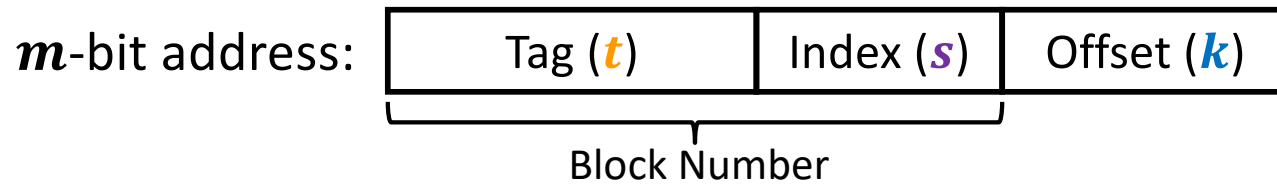
# Tags Differentiate Blocks in Same Index



# Checking for a Requested Address

- ❖ CPU sends address request for chunk of data
  - Address and requested data are not the same thing!
    - Analogy: your friend  $\neq$  his or her phone number

- ❖ TIO address breakdown:

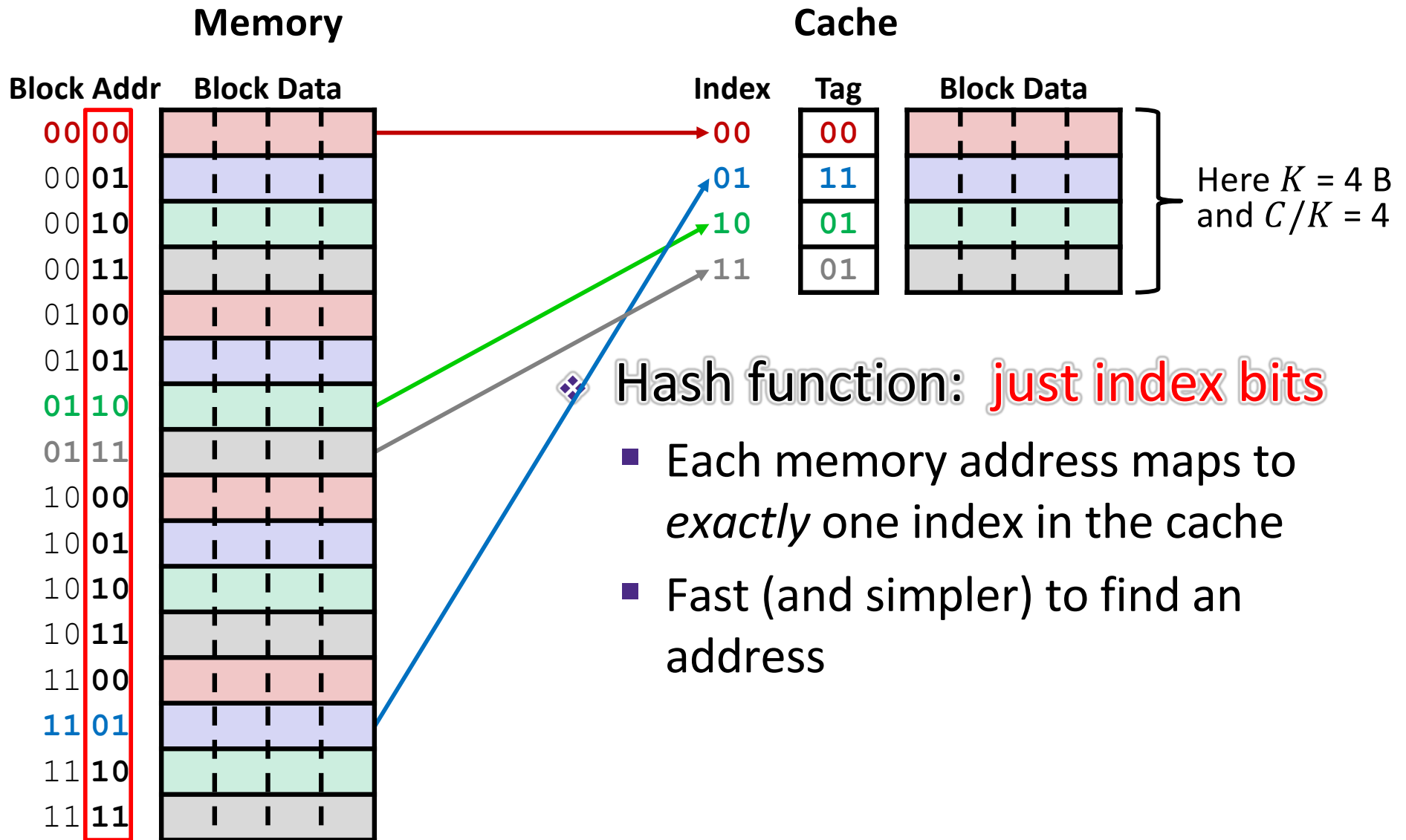


- **Index** field tells you where to look in cache
- **Tag** field lets you check that data is the block you want
- **Offset** field selects specified start byte within block
- **Note:**  $t$  and  $s$  sizes will change based on hash function

# Cache Puzzle

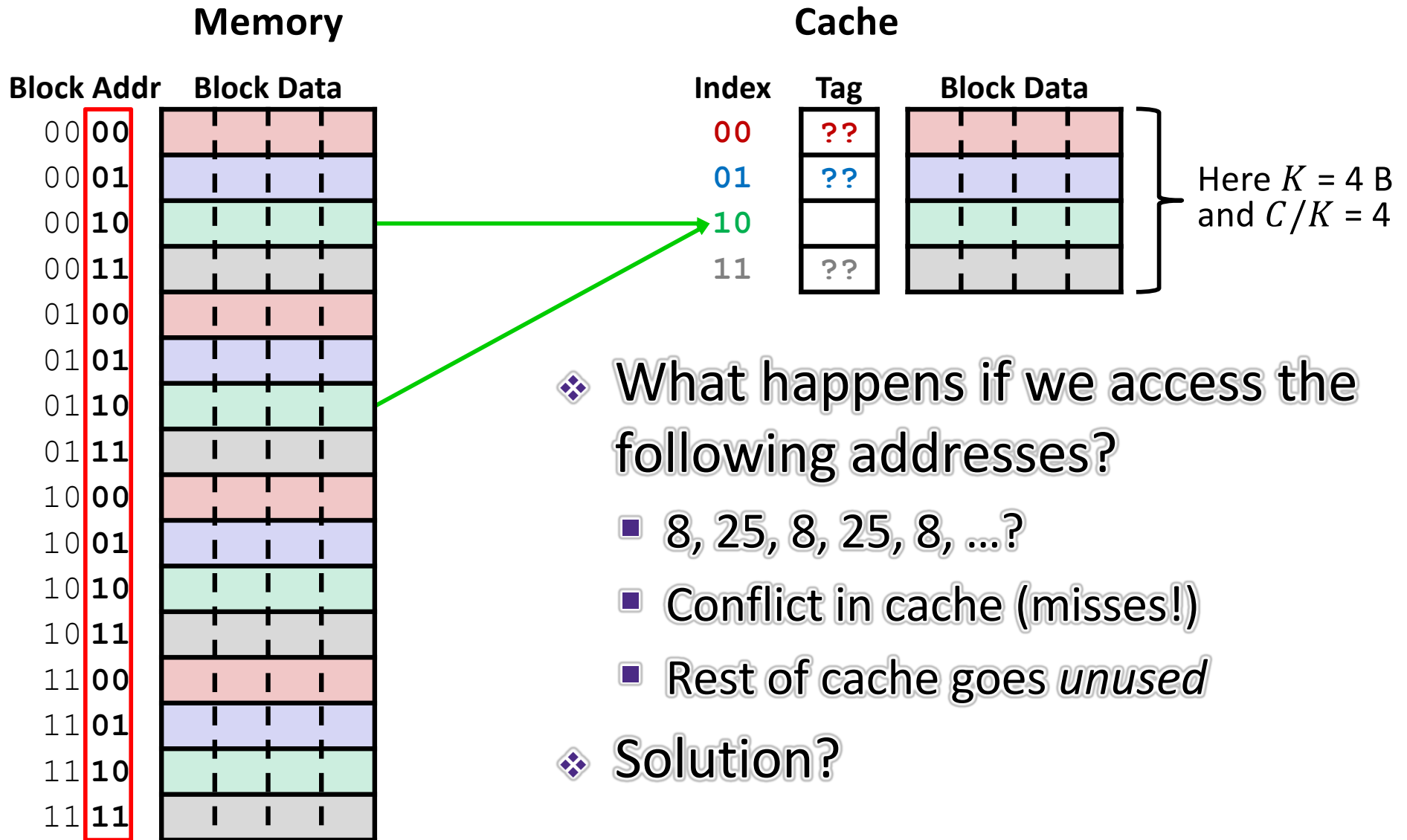
- ❖ Based on the following behavior, which of the following block sizes is NOT possible for our cache?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (14: miss), (15: hit), (16: miss)
  
- A. 4 bytes
- B. 8 bytes
- C. 16 bytes
- D. 32 bytes
- E. We're lost...

# Direct-Mapped Cache





# Direct-Mapped Cache Problem



- ❖ What happens if we access the following addresses?
  - 8, 25, 8, 25, 8, ...?
  - Conflict in cache (misses!)
  - Rest of cache goes *unused*
- ❖ Solution?

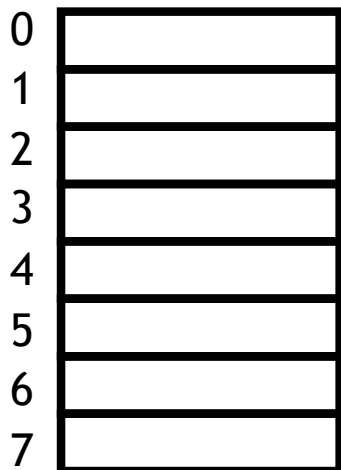
# Associativity

- ❖ What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- ❖ So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**

1-way:

8 sets,

1 block each

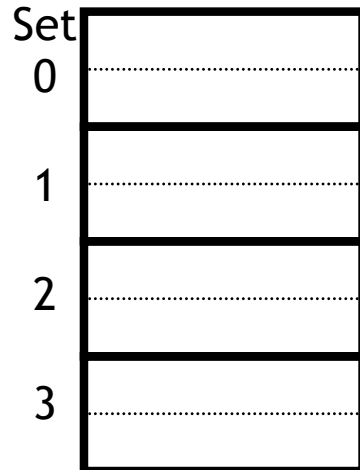


direct mapped

2-way:

4 sets,

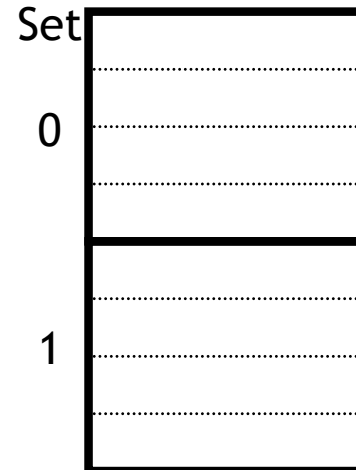
2 blocks each



4-way:

2 sets,

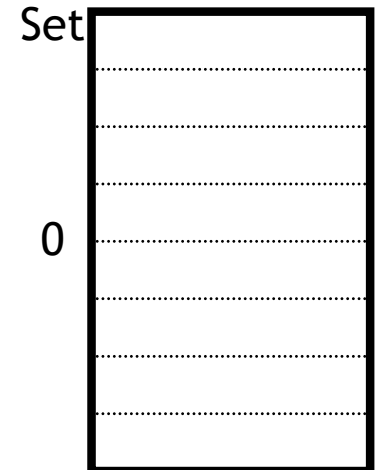
4 blocks each



8-way:

1 set,

8 blocks



fully associative